Performance and Scalability of Web Services Integration in the Plumtree Corporate Portal 4.5

Intel solution services

Project Final Report

Version 1.0

May 7, 2002

Prepared for: Plumtree Software

plumtree®
The Corporate Portal Market-Leader

intel

Table of Contents

Executive Summary

A corporate portal is designed to aggregate services securely from enterprise applications across the corporate network and the Internet. As corporate portals have grown in sophistication and ambition, the services they must aggregate have become more complex. A single portal page which several years ago might have contained links to company headlines now offers access to a company's most valuable electronic resources, including Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Content Management systems.

The popularity of portals coincides with and feeds on the development of Web services. Broadly speaking, a Web service is a programmatic component that is addressable remotely, typically using the Internet-standard HTTP protocol. Increasingly, the components that make up users' personalized portal pages will be generated by Web services. As the portal becomes a means by which end-users consume Web services, it becomes increasingly important to ensure that the portal can do so scalably and reliably.

The performance and scalability implications of aggregating many different services on a single page have not been rigorously explored. The foremost challenge is response time. Since users will not tolerate sluggish response times, a portal must deliver a page containing a large number of services in less than a second. Since the portal aggregates systems whose speed and reliability cannot be guaranteed, response times cannot be bottlenecked by slow services or network latencies. Since a mature corporate portal may integrate hundreds or even thousands of services, response time cannot degrade substantially as new services are added to the system. A portal that spends a great deal of time waiting for components to process will bottleneck as more users make requests, leading to serious scalability limitations.

The Plumtree Corporate Portal uses a messaging component known as the Parallel Portal Engine to execute Gadget Web Services (the company's term for remote services that include user interface components) in parallel. Plumtree Software claims that the Parallel Portal Engine is designed to avoid response time problems inherent in serial execution: each Web service will take some time to respond; if each is accessed serially, the response time of the aggregate page must be at least the total response time of each Web service. For example, if a user has 10 Web services displayed on a personalized page and each gadget takes one half second to deliver content, then the user will experience a minimum page response time of five seconds. With parallel execution, the same page should return – at minimum – at the rate of the slowest Web service, in this case one half second.

This test – the first independent test of Web services aggregation in a portal – was designed to determine whether a portal can consume increasing numbers of Web services without degrading performance. To conduct the test, the Plumtree Corporate Portal 4.5 and up to 1000 concurrently integrated Gadget Web Services were used. The number of Web services the portal consumed was varied in two ways: services were added to a page to determine whether aggregating additional services on a page negatively impacted response time, and the number of Web services concurrently utilized was also increased to ascertain whether the portal's performance degraded as the total number of services accessed by users increased.

It was determined that adding additional services to a personalized page increased the response time by only 11 milliseconds per additional Web service, meaning that users can add Gadget Web Services to their pages without experiencing appreciable performance degradation. It was also found that additions to the total number of services concurrently utilized did not appreciably

affect response time. In conclusion, the Plumtree Corporate Portal 4.5 aggregates Web services efficiently and scalably, in a manner that delivers consistent response time to end users.

Definitions

My Page: in the Plumtree Corporate Portal, a page composed of separate components – known as Gadget Web Services – that the user chooses. This is the most commonly accessed type of portal page.

Community Page: in the Plumtree Corporate Portal, a page shared by multiple portal users that is composed of Gadget Web Services.

Web service: a programmatic component accessible to other programs over standard internet protocols.

Gadget Web Service: a type of Web service specific to Plumtree Software that includes an HTML or XML user interface component and is designed to be aggregated directly into a user's My Page or a Community Page. Sometimes abbreviated as "gadget."

Gadget Server: in the Plumtree Corporate Portal, any server that outputs Gadget Web Services to be consumed by one or more portal servers.

Concurrently Accessed Gadget Web Services (CAG): the number of Gadget Web Services registered with the system that have been embedded in a My Page or Community Page for an active user.

Number of Gadgets per Page (NGP): the number of Gadget Web Services embedded in a particular My Page or Community Page.

Parallel Portal Engine (PPE): an HTTP-based messaging component of the Plumtree Corporate Portal that orchestrates the execution of multiple remote Web services concurrently, for eventual integration in a single Web page.

Vuser: in LoadRunner, a virtual user that simulates a browser and sends requests from it, generating load on the server.

Vuser Pacing: in LoadRunner, a runtime setting specifying the time interval between script iterations for each vuser.

Scenario: in LoadRunner, the scheduling and coordination of vuser activities.

Project Summary

Project Overview

This project focused on testing and quantifying the characteristics of the Plumtree Corporate Portal and its HTTP-based messaging component, the Parallel Portal Engine, for integrating increasing numbers of personalized Gadget Web Services into a portal deployment. Response time is the foremost challenge for a corporate portal deployment that integrates a large number of services on a single page. Since users will not tolerate sluggish response times, a portal must deliver a page containing a large number of services in less than a second. Since the portal aggregates systems whose speed and reliability cannot be guaranteed, response times cannot be bottlenecked by slow services or network latencies. Since a mature corporate portal may integrate hundreds or even thousands of services, response time cannot degrade substantially as new services are added to the system. Plumtree Software claims that its Parallel Engine solves these problems by orchestrating remote services for execution in parallel, but these claims have not been measured or independently verified, leaving an important question about portal scalability unanswered.[1] *An implicit goal in the project was thus to establish two new metrics for measuring and reporting the scalability of portals: the number of services integrated and in use by portal users with negligible performance impact, and the number of dynamic services which can be accessed by one personalized page with negligible performance impact*.

Project Goals

The business and strategic objective for the *Plumtree Corporate Portal 4.5* Web services benchmark was to demonstrate the scalability and performance of parallel orchestration of remote services in general, and of Plumtree's Parallel Portal Engine in particular.

The technical objective of this project was focused on benchmarking performance characteristics of a single *Plumtree Corporate Portal 4.5* hardware configuration and systematically changing the number of Gadget Web Services in use by the system as well as the number of Gadget Web Services accessed concurrently per user. The main objectives were to:

- Determine the baseline performance characteristics of a portal system with no Gadget Web Services deployed and a large number of users;
- Determine the performance characteristics of the same portal system with many Gadget Web Services deployed when different numbers of Gadget Web Services were accessed by users;

[1] Two white papers have previously been published on the scalability of the Plumtree Corporate Portal. The first, published by Dell Corporation's Application Solutions Center, established that hits per second is the proper metric for characterizing load on a portal, and that the Plumtree Corporate Portal scales linearly when additional Web servers are added to a server farm running Dell's Intel-based servers. The second, published by the Microsoft Technology Center, established that the Plumtree Corporate Portal's database, running SQL Server 2000 on Intel OEM hardware, could accommodate one million named users with acceptable performance.

- Determine the baseline performance characteristics of a portal system with a large number of users, each of whom accesses one Gadget Web Service on his personalized page; and

- Determine the performance characteristics of the same portal system with the same number of users, each of whom accesses different numbers of Gadget Web Services on his personalized page.

These goals were measured by showing acceptable response time dependence as well as system stability (processor utilization, network utilization, available memory, and other relevant system resources) under sustained load as test parameters were varied.

Non Goals

Since multiple Web server scale-out and database scale-up to hundreds of thousands of users have been demonstrated in earlier tests, we focused this test explicitly on testing scaling up the number of Web services integrated into the portal environment. Readers interested in metrics such as the number of hits a portal Web server can sustain or the number of users a portal can serve should consult earlier studies.

Testing Environment

High-Level Overview

The *Plumtree Corporate Portal 4.5* server farm environment was duplicated in the Intel Solution Services Integration Lab located in Santa Clara, CA. Figure 1 is a high-level illustration of the testing environment.

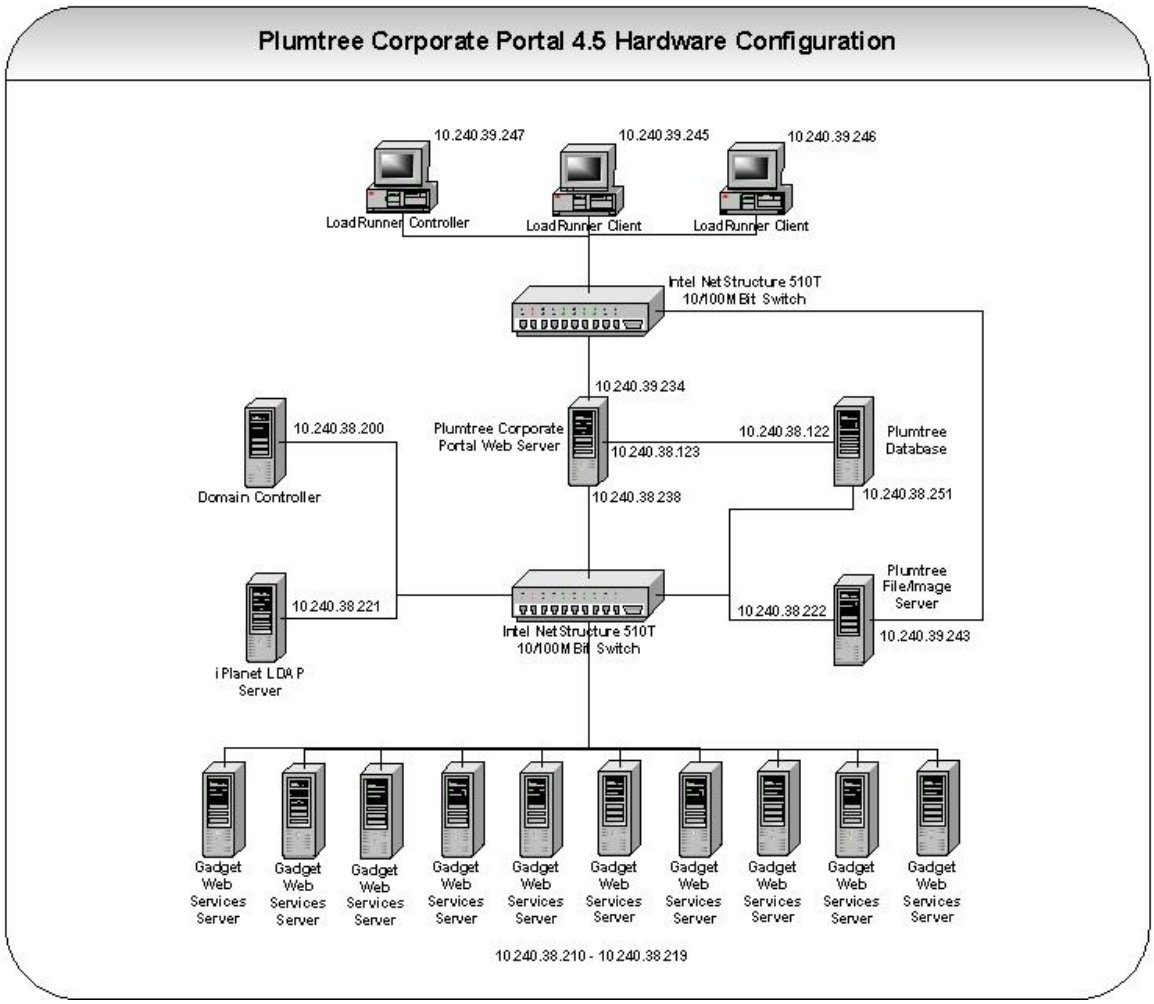

Figure 1: Overview of Testing Environment

Hardware and Software Configuration

The *Plumtree Corporate Portal 4.5* server farm was implemented in a testing environment comprising eighteen Intel Architecture-based servers. Although generic Intel Architecture based servers were used during testing, similar results presented in this document were expected if comparable OEM servers were implemented.

Two servers acted as load generation clients and were required to produce the necessary workload applied to the *Plumtree Corporate Portal 4.5* server. Another server acted as the controller for the load generation clients and captured *Microsoft Windows Performance Monitor* data. The following are hardware and software details for these servers:

- Hardware
  - Four Intel Pentium III Xeon 700 MHz processors
  - 4 GB SDRAM
  - (2) 9 GB SCSI Hard Disk
  - (1) 10/100 Network Interface Card
- Software
  - *Microsoft Windows 2000 Advanced Server* with Service Pack 2
  - *Mercury Interactive LoadRunner 7.5*

One server was used as the Web server for the *Plumtree Corporate Portal 4.5* Web application.[2] The following are details for the Web server:

- Hardware
  - Four Intel Pentium III Xeon 700 MHz processors
  - 1.5 GB SDRAM
  - (2) 18.2 GB SCSI Hard Disk
- Software
  - *Microsoft Windows 2000 Advanced Server* with Service Pack 2
  - *Plumtree Corporate Portal 4.5 Web Application*

One server was used as the database server running *Microsoft SQL Server 2000 Enterprise Edition*. The following are details for the database server:

- Hardware
  - Eight-processor
  - Intel Pentium III Xeon 700 MHz
  - 8 GB SDRAM
  - (2) 9 GB SCSI Hard Disk
- Software
  - *Microsoft Windows 2000 Advanced Server* with Service Pack 2
  - *Microsoft SQL Server 2000 Enterprise Edition* with Service Pack 2

[2] Since scale-out and the total sustainable throughput of the server farm were not at issue, we did not deploy load-balanced Plumtree Portal Servers. Earlier tests demonstrate linear scalability in the presence of load-balanced portal servers.

One server was used as the Plumtree File/Image server. The following are details for the File/Image server:

- Hardware
  - Two Intel Pentium III Xeon 700 MHz processors
  - 512 MB SDRAM
  - (2) 9 GB SCSI Hard Disk
- Software
  - *Microsoft Windows 2000 Advanced Server* with Service Pack 2
  - *Plumtree Shared Information Server*

One server was used as the LDAP server running *iPlanet Directory Server v4.13*. The following are details for the LDAP server:

- Hardware
  - Two Intel Pentium III Xeon 700 MHz processors
  - 512 MB SDRAM
  - (2) 9 GB SCSI Hard Disk
- Software
  - *Microsoft Windows 2000 Advanced Server* with Service Pack 2
  - *iPlanet Directory Server v4.13*

Lastly, ten systems were used as generic Gadget Servers. The following are details for these servers:

- Hardware
  - One Intel Pentium III 933 MHz Processor
  - 256 MB SDRAM
  - (1) 9 GB IDE Hard Disk
- Software
  Microsoft Windows 2000 Advanced Server with Service Pack 2

Software Logical Configuration

In this test, there are three principal variables to the software configuration: the Plumtree Portal Web Server and operating system tuning parameters, the Plumtree database logical configuration, and the characteristics of the remote Gadget Web Services being integrated into the portal.

Web server and Operating System Tuning Parameters

Operating System Tuning

- **Network Service Optimization:**
  - Location on *Microsoft Windows 2000 Advanced Server*: Right Click – Properties on "My Network Places" >> Right Click – Properties on the particular network interface >> Properties on File and Printer Sharing for Microsoft Networks
    - For Databases, Web Servers, and Job Servers, set this to ***Maximize data throughput for network applications***
    - For the Shared Files, set this to ***Maximize data throughput for file sharing***
- **Disable Foreground Application Performance Boost**
  - *Microsoft Windows 2000 Advanced Server*: System Properties Control Panel >> Advanced >> Performance Options. Set it to optimize performance for ***Background services***
- **NIC Receive Buffers** (this may not exist on your system)
  - *Microsoft Windows 2000 Advanced Server*: Right Click – Properties on "My Network Places" >> Right Click – Properties on the particular network interface >> Configure >> Advanced >> Receive Buffers. Set to ***256***
- **MaxUserPort**
  - Add registry value **MaxUserPort**, if it's not already there, to HKLM\System\CurrentControlSet\Services\TcpIp\Parameters and set it to **0xfffe** (65534)

IIS Tuning

General Tuning:

- Enable **Buffering** for ASP Applications in the IIS MMC. (Plumtree Virtual Directory Properties >> Virtual Directory >> Configuration >> App Options)
- IIS Web-Site Performance Bar set to **More Than 100,000** (IIS MMC >> Right Click – Properties on the Web Site containing the Plumtree Virtual Directory >> Performance Tab)
- Set **ASPProcessorThreadMax** (*Microsoft Windows 2000 Advanced Server*). In general, this does not need to be changed for *Microsoft Windows 2000 Advanced Server* (it defaults to a value of 25).
- Set **AspScriptEngineCacheMax** to the value of **ProcessorThreadMax** multiplied by the number of processors in the system plus one. For instance, if there are four processors and **ProcessorThreadMax** is set to 25, then it should be set to 125 = [25 * (4 +1)]. This is set through the *Number of Script Engines Cached* setting (in IIS MMC >> Plumtree Virtual Directory properties >> Virtual Directory >> Configuration >> Process Options)
- Set the **Script File Cache** to ***Cache all requested ASP files*** (IIS MMC >> Plumtree Virtual Directory properties >> Virtual Directory >> Configuration >> Process Options)
- The Indexing service is turned off. Additionally, turn off Windows Indexing of the virtual directory (IIS MMC >> Plumtree Virtual Directory properties >> Virtual Directory >> Uncheck "Index this resource").

Scalability Testing Specific Tuning:

- Set the **Session Timeout** to be 120 minutes. (This prevents sessions from timing out during one test run.)

Plumtree Database

The Plumtree database was configured before testing began. The database was populated to simulate an environment representative of a production portal system. There were 19000 distinct users in the system, divided into 19 groups of 1000 users each. Each group was imported from, and authenticated to, an iPlanet LDAP system. There were 3805 remote Gadget Web Services registered in the system, 19000 distinct My Pages, 1010 distinct communities, and 11415 distinct sets of user preferences.

Each user group was used for different tests and included users with different profiles. Each user had one My Page. Each user was in one community that had the same profile as the user's My Page. Users in each group had a specific number of gadgets on their My Pages and community pages. Each group used a specific number of gadgets in total. This configuration was created by using a specific number of distinct profiles for each group. This was done to keep the database configuration constant through the tests, thereby making the results comparable because database changes are not a factor. Table 1 below describes the user profiles of the different groups.

Group Name	Number of Gadgets per My Page and Community Page	Total Number of Gadgets Utilized by Group	Number of Distinct User Profiles in Group
01	1	1	1
02	2	2	1
03	3	3	1
04	4	4	1
05	5	5	1
06	6	6	1
07	7	7	1
08	8	8	1
09	9	9	1
10	10	10	1
02_50	2	50	25
02_200	2	200	100
02_1000	2	1000	500
05_50	5	50	10
05_200	5	200	40
05_1000	5	1000	200
10_50	10	50	5
10_200	10	200	20
10_1000	10	1000	100

Table 1: User Group Configuration

The first column in the table is the name of the user group. The second column shows the number of Gadget Web Services that each member of the group has on both his My Page and his community page. The third column shows the total number of distinct Gadget Web Services utilized by members of the group. The fourth column shows the number of distinct user profiles used in that group.

There were 10 gadget server objects in the system. The number of a gadget on the page determines which gadget server object the Gadget Web Service utilizes. For example, the first gadget on a user's page utilizes gadget server object 1, and the second gadget on a user's page utilizes gadget server object 2. Each gadget server object represented a different physical gadget server computer.

To simulate the most demanding possible conditions, in most tests no Gadget Web Services were cached by the portal; all content is fresh.[3] Caching was eliminated to ensure that each Gadget Web Service actually had to execute – and thereby introduce latency -- to generate a result. Had caching been employed, it would have been impossible to determine the efficacy of the parallel processing architecture, since some services would effectively generate their content instantaneously. This also models dynamic Web services that may have their own caching, independent of the portal. For comparative purposes, some background tests using different cache settings were run by setting different Gadget Web Services to use caching. See Appendix A for a discussion of these tests.

Remote Gadget Web Services

A remote Web service (or Web service generally) is remote application logic accessible via standard Web protocols. A remote Gadget Web Service is a Web service that returns user-facing content to be aggregated in a personalized portal page. In other words, Gadget Web Services are designed for user interaction and typically contain an HTML-based user interface[4]. The tests conducted utilize a remote Gadget Web Services model. The Gadget Web Services are categorized using the following quantities:

1. **Average Web Service Latency (AWSL).** AWSL is the average time from the last byte of the service request sent to the last byte received of the Web service response. It will be a combination of the time the Web service takes to process on the remote server and the time the request from and response to the portal server takes to travel over the network.
2. **Average Web Service Response Size (AWSRS).** AWSRS is the average size, in bytes, of the Web service request response.
3. **Average Number of Personalization Preferences (ANPP).** ANPP is the average number of personalization preferences sent to the Web service as part of a request. Gadget Web Services typically contain personalization preferences that determine how the Web service presents itself to end users. Preferences may be scoped for a single user, a community, or the portal as a whole. Examples of possible preferences are user authentication information or output format preferences.

Note that there are many other variables to consider when a Web service is integrated into the portal, including: language used to code the Web service, operating system and runtime

[3] The Plumtree Corporate Portal contains built-in caching algorithms. In most cases, Plumtree recommends that caching be used to improve performance.

[4] Gadget Web Services may contain other types of user interfaces, including XML. These are used much less often than HTML in production because it is typically more scalable to accomplish data transformations remotely on the Gadget Web server.

environment of the Web service, processing load on the Web service's host computer, type of user preference, and so on. These parameters are not relevant to this test; the quantities used to characterize Gadget Web Services are sufficient to model performance characteristics. Additionally, latency in the tests was normalized to simulate realistically the total latency that could accrue from these variables.

The test configuration models typical, realistic aggregations of remote Web services for corporate portals. To accomplish this goal, we can envision several different classes of Gadget Web Services:

1. **Enterprise Class Web Service Frameworks (ECWSF).** These Web services are highly configurable services that integrate enterprise applications such as e-mail and CRM systems. They typically have multiple user preferences.
2. **Legacy System Integration Web Services (LSIWS).** These Web services present a Web interface for legacy applications or data warehouses. They require less computation than Enterprise Class Web service Frameworks.
3. **Simple Database Web Services (SDWS).** These Web services present a Web interface for simple database queries, such as personalized links or simple poll results.
4. **Simple Web Services (SWS).** These Web services present basic information such as weather, stock quotes, news feeds, welcome messages, announcements, and screen-scraped intranet sites.

The Average Web service Latency characteristic can be discussed in light of the above functional classes of Web services. A model Web service with a large latency could be an Enterprise Class Web service Framework or legacy system integration Web service, which perform complex tasks or integrate with slow back-end systems. Simple Web services on slow networks could also cause large latencies; for example, a news feed might be transmitted over the Internet or a slow WAN. Latency could also be introduced by a simple database Web service with a slow backend database or a slow intranet connection; for example, a database with a billion rows might return even simple queries rather slowly.

Any realistic test of Web services aggregation must take into account Web service latency. As the above explanation demonstrates, a Web services architecture must contend with many different sources of latency. Cached or zero-latency services would not create a realistic test scenario for the integration of enterprise applications or distributed Web services.

The Gadget Web Service used in this test, the WaitGadget, is a model representing an average Gadget Web Service in use. It is characterized by the following parameters:

1. The AWSL is 500 milliseconds, with a variation between 475 and 525 milliseconds
2. The AWSRS is 2 KB.
3. The ANPP is 3; each user has three preferences for each gadget.
4. No Portal caching is used; fresh content is delivered for every user request.

The response time of 500 milliseconds is reasonable for complex Gadget Web Services or services accessed via a slow connection. The response size of 2 KB is reasonable for Gadget Web Services with formatting and javascript. The number of preferences used is typical: one preference for identification or authentication, one preference for specifying data location or repository, and one preference for specifying formatting information.

The WaitGadget used was written in Active Server Pages using a COM component to generate the delay. The programming language and runtime environment are inconsequential in this test; the Web service might have been written in Java or C#.

Testing Methodology

Architecture Assessment Methodology

Intel Solution Services implements a straightforward methodology for architecture assessment and is illustrated in Figure 2.

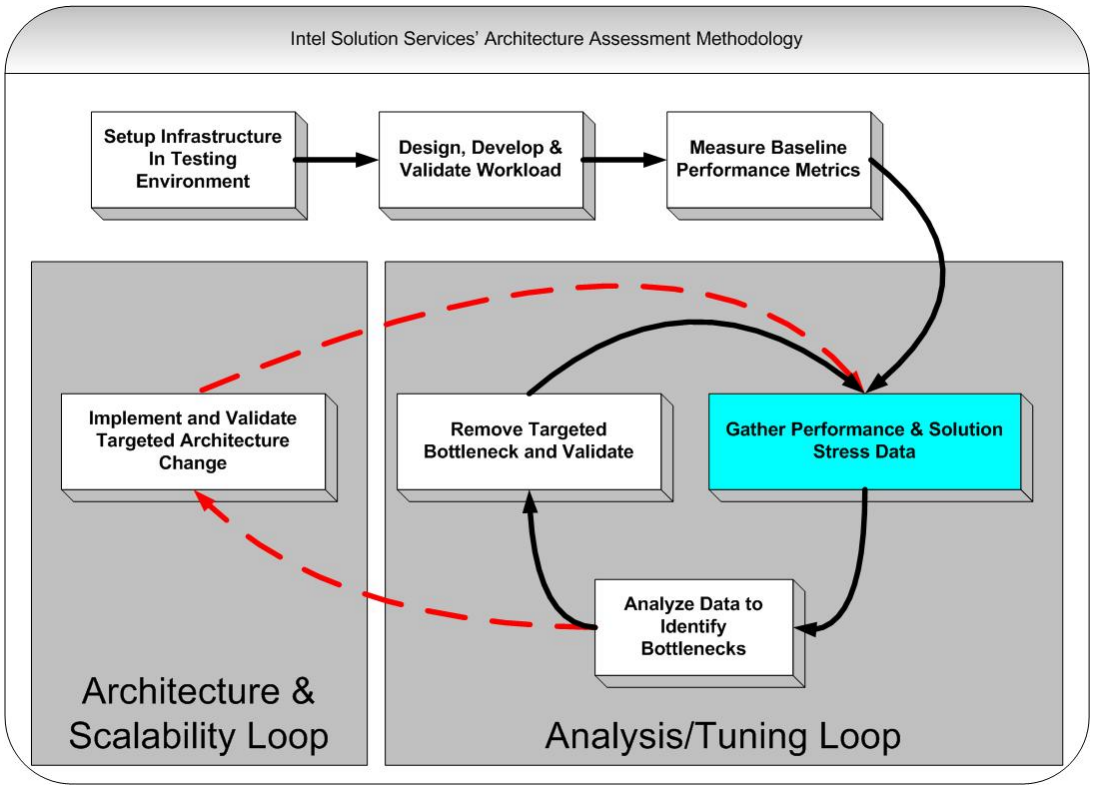

Figure 2: Intel Solution Services' Architecture Assessment Methodology

After the infrastructure was setup and validated in the Santa Clara Intel Solutions Center, the Plumtree team worked to research, design, develop, and validate a workload to use for stress testing the *Plumtree Corporate Portal 4.5* server farm configuration (details are presented under *Workload Design Methodology* in this section). Once an appropriate workload was validated, baseline performance metrics were gathered. After baseline metrics were gathered, an iterative process of gathering data, analyzing the data, modifying the solution, and validating the modification was completed several times (all testing data are presented in the *Testing Results* section of this document).

This methodology helped ensure that all architecture changes made to the solution were validated and that the exact performance impact of each modification was recorded.

Performance Workload Design Methodology

The Plumtree and Intel Solution Services team worked together to research, design, develop, and validate an appropriate workload that satisfied four requirements:

- Representative;
- Measurable;
- Static; and
- Repeatable.

The most important characteristic of a workload is that it must be representative. This means that the workload must simulate a load that closely mimics that found in the production environment. Secondly, the workload must be measurable which implies that it allows metrics to be measured during the application of the workload. Next, the workload must be static, which means the workload is stable enough to allow the solution to be measured. Lastly, the workload must be repeatable. A repeatable workload implies that successive test runs will apply the exact same load to the solution infrastructure. This allows for accurate comparison of metrics between these runs, which is critical for determining the effects of tuning and architecture modifications.

LoadRunner 7.5 with 1000 virtual users, all running the same script, was used to generate the load. The configuration section above specifies the distribution of client machines and the network configuration. The virtual users were distributed evenly over the client machines.

Two test users were configured to validate the absence of specific errors: a general portal or Gadget Web Service error, and a timeout error. None of these errors was encountered during test runs. Response code errors and Web server errors were monitored and logged automatically by the tool. None of these errors was encountered during test runs.

For the purposes of this test, we focused on My Pages and Community Pages, not searches or browsing of the portal's Document Directory, since My Pages and Community Pages integrate Gadget Web Services while search and browsing do not. LoadRunner virtual users (vusers) were initialized before testing began by logging in to the portal as different portal users in the user group relevant for the test. To avoid the creation of an unrealistic bottleneck from the LoadRunner testing scenario, no more than 20 users were initialized at once. Runtime settings were set so that vusers ignored think times, so that vuser pacing determined the load.

The action of the script was a refresh of a personalized page, either the My Page or the Community Page, whichever was relevant for the test run. The LoadRunner script used is included in Appendix B at the end of this document. The number of users actively running was ramped up at a rate of 25 users every 15 seconds until all 1000 vusers were accessing the system. The LoadRunner testing script was run repeatedly for 20 minutes after ramp-up was complete. The pacing of the vusers refreshing their personalized page was set at between 42 to 47 seconds each iteration. This leads to an average load of 22.5 hits/sec on the portal, with a variation of 0.3 hits/sec during steady state.

The script used models a real world scenario as well as isolating the system's performance based on responses to a particular user action. Users are expected to refresh their personalized page repeatedly to view updated content and to utilize interactive Gadget Web Service functionality. Additionally, many users employ the portal as a Web desktop, setting their portal pages to refresh automatically in the background after a fixed amount of time.

The load generated is typical of average peak loads per server for real deployments. To ensure stability and allow for extreme spikes, system architects typically plan for about 30 to 40 percent processor utilization at peak loads, depending on particular company policy.

The case tested was constructed so that the bottleneck in performance is the portal server rather than the network or the database. Given fixed load and configuration, as well as a high-end database and fast 100 Base-T network, the latencies due to network lag and database interaction are constant across tests, and negligible. By subtracting a baseline latency from test results, the effects of these factors on the data is removed. The absolute response time data shows that these factors account for at most 100 milliseconds, out of the baseline response time with one WaitGadget included.[5] The network delays were measured to be below 10 milliseconds and the database response times were under 20 milliseconds during calibration runs. There are two database calls per page, so this accounts for about 50 milliseconds of latency.

To ensure a controlled test, the Internet Information Server was restarted between each testing run.

Performance Metrics and Data Gathered

The following are a list of the unique pages that were requested in the workload and reflect the transaction names within the *LoadRunner* script:

- My Page
- Community Page

Three key performance metrics were extracted from the *LoadRunner* data for each of these pages/transactions:

- Response time of transaction
- Processor utilization on the portal server
- Hits/sec on the portal server

Each scenario was monitored for errors, and none produced any errors during data collection.

Data was collected and averaged at a static load between 18 and 27 minutes into the test. This is approximately 5 minutes after the completion of ramp-up.

Remote Gadget Web Services per Page Latency Metric (WSPL)

Web Services per Page Latency (WSPL) is defined as the average response time increase of a personalized portal page that integrates and aggregates multiple remote Gadget Web Services (which were defined in the Software Configuration section of this document) as the number of Gadget Web Services increases from one to ten. The system on which WSPL is measured must use the hardware specified in the Hardware Configuration section. The scenario used to measure response time of the personalized portal page must be the scenario specified in the Performance Workload Design Methodology section.

[5] Of interest, but not directly relevant to the test, is the baseline response time with the WaitGadget cached – in other words, without the latency created by the WaitGadget. We found this response time to be 83 milliseconds.

The implicit baseline definition is the response time of a personalized portal page integrating one remote Gadget Web Service.

WSPL has units of milliseconds per Gadget Web Service integrated.

If the increase in portal personalized page response time per additional remote Gadget Web Service integrated is linear, then WSPL is the slope of the graph of the difference from baseline response time of the portal personalized page response time versus the number of Gadget Web Services integrated minus one, from zero to nine. If the graph is non-linear, then WSPL may have strong dependence on the number of Gadget Web Services aggregated, but can be computed.

Web Services per Page Latency is a key metric characterizing the scalability and performance of a portal. A loosely coupled Web services architecture has clear advantages for portal system design; because remote Web service code is isolated from portal computers, problems with remote Web services do not affect the portal computers directly, remote Web services do not consume portal computer resources, and software interaction incompatibilities which could arise if multiple Web services were co-located are avoided.

However, the loose coupling of remote Gadget Web Services poses the technical problem of portal response time. The portal must be able to integrate and aggregate remote Gadget Web Services located in different networks and taking an unknown time to respond to requests. If the portal does not integrate and aggregate remote Gadget Web Services effectively, these network and process delays could all add up to make portal personalized page response time unacceptable for users. Eventually, the portal will queue up such a large volume of requests that its ability to scale to a large number of end-users will be compromised.

WSPL is defined to cleanly characterize the performance cost of integrating dynamic remote Gadget Web Services which deliver fresh content and application access to the user. These types of Web services are the most useful to business users, and so are likely to be the most widely utilized in a real system. Examples of remote Gadget Web Services in this class are e-mail inboxes and discussion databases. The Web services will have internal caching, but the portal cannot cache content from these types of services without compromising utility unacceptably.

Fixing the hardware used and load applied isolates performance measurements from direct scale measurements. The system is designed so that the portal server latency should be the bottleneck for any portal tested, and the tests are designed to create a baseline from which we can compare performance as additional services are added to the system. By fixing the scale – number of users and number of hits per second – to create a baseline, and ensuring that network or other external latencies will not bottleneck the system and are constant across test runs, we have ensured that the result is generally applicable, reproducible, and comparable.

The tests run calculate WSPL for the Plumtree Portal My Page by using tests 01, 02, 03, 04, 05, 06, 07, 08, 09, and 10 to measure Portal My Page response times for My Pages integrating one to ten remote Gadget Web Services. The Plumtree Portal Community page was also tested, to show that its behavior is comparable to the My Page WSPL.

Remote Gadget Web Service Concurrently Accessed Latency Metric (WSCAL[n])

Web services Concurrently Accessed Latency (WSCAL[*n*]) is defined as the average response time increase of a personalized portal page with *n* aggregated Gadget Web Services where *n* is greater than one, as the number of Gadget Web Services concurrently utilized by the portal increases from fifty to one thousand. The system on which WSCAL[*n*] is measured must use the hardware specified in the Hardware Configuration section. The scenario used to measure

response time of the personalized portal page is the scenario specified in the Performance Workload Design Methodology section.

The implicit baseline definition is the response time of a personalized portal page with *n* aggregated remote gadgets with fifty remote gadgets concurrently utilized.

WSCAL[*n*] has units of milliseconds per remote Gadget Web Services concurrently utilized.

An important metric of portal scalability is the system's ability to scale out the total number of remote Web services utilized without response time degradation. Response times unacceptable to users could arise because of technical scalability difficulties, such as excess processor utilization on the portal server in managing many more Web service connections, or technical latency problems, such as inefficient processing of Web service interactions. The total number of Web services registered with the system – but not concurrently utilized by users – could affect the latency, but this is likely to be a data layer problem only; database calls can be optimized to remove this bottleneck. Because WSCAL[*n*] measures Web services being concurrently utilized by users, it targets the main risk area of portal page response times. It is thus the key metric for demonstrating portal scalability as remote Web services are added to the system.

Showing that the WSCAL measurements for different values of n is the same validates that the metric is independent of the number of remote Gadget Web Services aggregated per portal page. If WSCAL[*n*] has significant dependence on the value of *n*, the portal cannot truly scale for large numbers of concurrently accessed Web services, because different users can choose personal settings which cause unacceptable portal response times.

Fixing the hardware, load, and number of aggregated remote Gadget Web Services per page isolates the dependence on the number of remote Gadget Web Services concurrently accessed.

The tests run measure WSCAL[*n*] for the Plumtree Portal My Page for *n* values of 2, 5, and 10. Tests 02_50, 02_200, 02_1000, 05_50, 05_200, 05_1000, 10_50, 10_200, and 10_1000 were used. Tests measuring WSCAL[10] for the Plumtree Portal Community page, using tests 10_50, 10_200, and 10_1000, were run to validate that Community Page measurements are comparable to My Page measurements.

Load Dependence of Portal Performance

The metrics collected require a fixed load that models real world average peak load per portal server to remove load dependence from the performance metrics. To ensure that the tests were generally applicable, tests were conducted to determine the sensitivity of portal performance to changes in applied load. A modified version of the scenario described previously which ramped up vusers at the same rate but had a much faster user pacing – thereby generating more hits per second – was run, to the point where the portal server's processor utilization was saturated. As long as the portal server is utilized below the saturation level, performance does not change dramatically at different load levels; it changes at most a few hundred milliseconds, and for loads utilizing less than 70% processor utilization, it changes less than 100 milliseconds. Tests 01 and 10 were run with this faster user pacing.

Testing Results & Analysis

All the results from the testing of the *Plumtree Corporate Portal 4.5* server farm configuration are presented in this section. Detailed descriptions of each architecture change are followed by the performance metrics gathered and listed in the *Testing Methodology: Performance Metrics and Data Gathered* section of this document.

Tests Varying the Number of Gadget Web Services per My Page

Average gadget response time was set to 500 milliseconds. The response time measured for a test run is the base 500 milliseconds for the Gadget Web Service plus a baseline overhead of the personalized page plus the latency due to the number of Gadget Web Services aggregated per page. Using the response time of a personalized page with one Gadget Web Service as a baseline isolates the performance changes due to the aggregation of additional Gadget Web Services. This baseline response time was measured to be 582 milliseconds.[6] The standard deviation of this mean was computed to be 2 milliseconds.

Table 2 below shows the data collected for each test run. Figure 3 on the following page shows the latency changes from the baseline as the number of Gadget Web Services aggregated per page is changed.

Test Name (Number of Gadgets per My Page)	Response Time (ms)	Response Time Difference From Baseline (ms)	% Total Processor Utilization
01	582 +/- 2	0	23%
02	592 +/- 1	10 +/- 2	25%
03	605 +/- 1	23 +/- 2	28%
04	618 +/- 1	35 +/- 2	31%
05	628 +/- 1	46 +/- 2	31%
06	640 +/- 1	58 +/- 2	36%
07	650 +/- 1	67 +/- 2	38%
08	660 +/- 2	77 +/- 3	42%
09	668 +/- 2	86 +/- 3	48%
10	678 +/- 2	95 +/- 3	45%

Table 2: Data for Tests Varying Number of Web services per My Page

Error estimates are the standard deviation of the mean. The load (hits/sec) on the portal server is kept constant at 22.5. A distribution of 0.3 hits/sec is built into the vuser pacing, and the standard deviation of the mean is measured to be within 0.2 hits/sec.

[6] It is important that this figure *not* be understood as the minimum response time for a personalized page. The 500 milliseconds latency built into the single Gadget Web Service on the page means that 500 milliseconds is the theoretical fastest response time for this page.

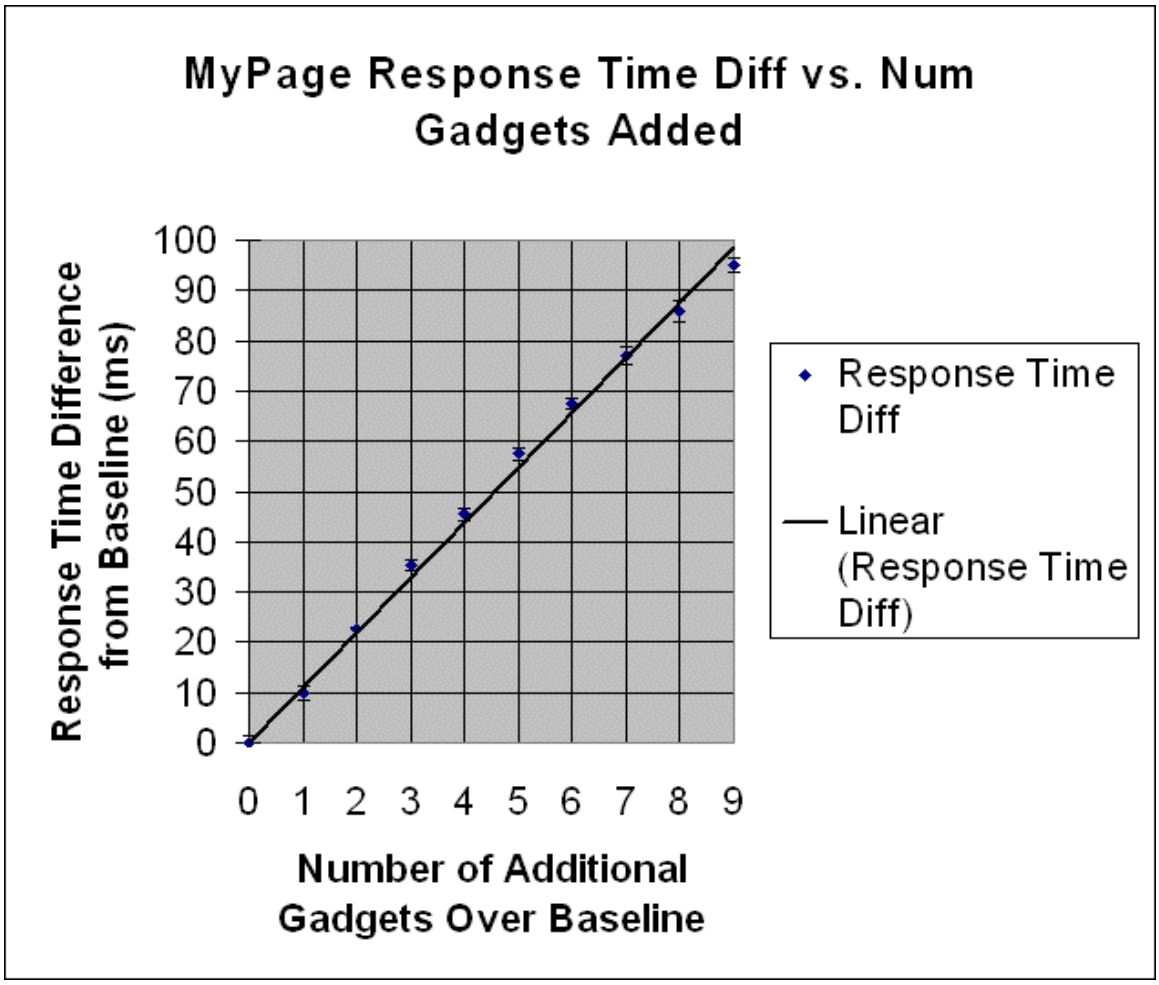

Figure 3: Response Time Differences in Milliseconds vs. Number of Gadget Web Services per My Page over Baseline of One Gadget

An X-value of 0 means there was one gadget on the My Page; this is the baseline. An X-value of 9 means that there were 10 gadgets on the My Page. Also shown is the graph of the best linear fit (least squares) to the data. The r-squared value for the fit is 0.9961. The slope of the best-fit line is 10.6 +/- 0.3 ms/gadget.

WSPL is calculated to be 10.6 +/ 0.3 ms per gadget web service integrated.

The data shows that the performance impact of adding another gadget to a personalized page is constant and negligible. A constant performance impact per additional gadget added means that the My Page can be personalized arbitrarily by users with predictable, stable results that scale for large numbers of gadgets per page. A performance impact of 10.6 milliseconds per aggregated gadget is negligible to users because users will not be aware of response changes less than a few hundred milliseconds.

Additionally, the absolute response times show that the performance overhead of the My Page is negligible to end users. Subtracting the 500 milliseconds theoretical minimum response time of

the My Page from the measured response times shows that the latency due to the My Page and the network total less than 200 milliseconds for all measurements.

Tests Varying the Number of Gadget Web Services per Community Page

Average gadget response time was set to 500 milliseconds. The response time measured for a test run is the base 500 milliseconds for the Gadget Web Service plus a baseline overhead of the personalized page plus the latency due to the number of Gadget Web Services accessed. Using the response time of a personalized page with one Gadget Web Service as a baseline isolates the performance changes due to the aggregation of additional gadget web services. This baseline response time was measured to be 622 milliseconds.[7] The standard deviation of this mean was computed to be 1 millisecond.

Table 3 below shows the data collected for each test run. This table shows the latency changes from the baseline as the number of Gadget Web Services aggregated per page is changed. Figure 3 on the following page shows a graph of the response time difference data for the Community Page.

Test Name (Number of Gadgets per Community Page)	Response Time (ms)	Response Time Difference From Baseline (ms)	% Total Processor Utilization
01	622 +/- 1	0	32%
02	636 +/- 2	14+/- 2	36%
05	674 +/- 2	53 +/- 1	40%
10	731 +/- 1	109 +/- 3	59%

Table 3: Data for Tests Varying Number of Web services per Community Page

Error estimates are the standard deviation of the mean. The load (hits/sec) on the portal server is kept constant at 22.5. A distribution of 0.3 hits/sec is built into the vuser pacing, and the standard deviation of the mean is measured to be within 0.2 hits/sec.

[7] It is important that this figure *not* be understood as the minimum response time for a personalized page. The 500 milliseconds latency built into the single Gadget Web Service on the page means that 500 milliseconds is the theoretical fastest response time for this page.

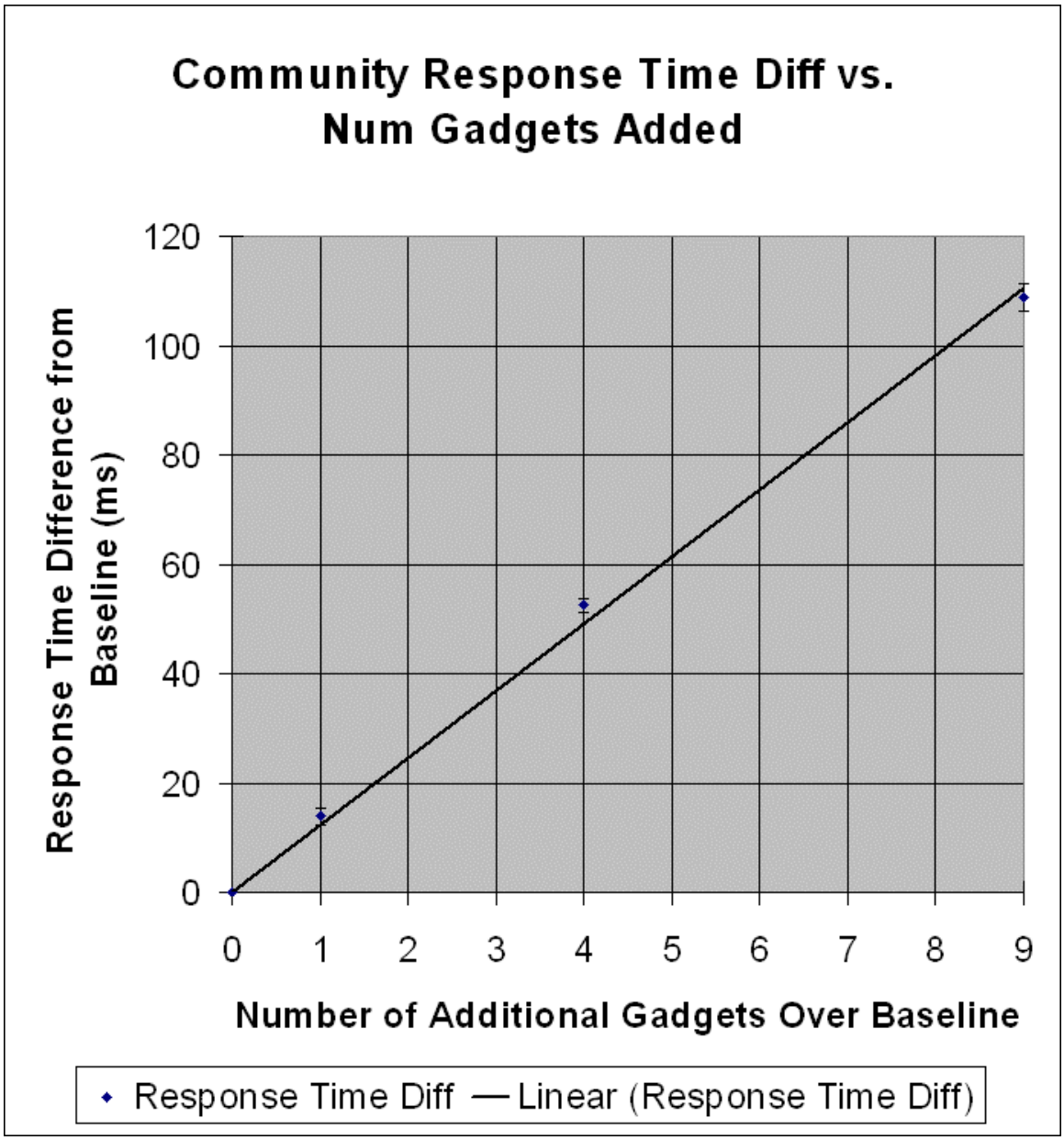

Figure 4: Response Time Differences in Milliseconds vs. Number of Gadget Web Services per Community Page over Baseline of One Gadget

An X-value of 0 means there was one Gadget Web Service on the Community Page; this is the baseline. An X-value of 9 means that there were 10 gadgets on the Community Page. Also shown is the graph of the best linear fit (least squares) to the data. The r-squared value for the best-fit line is 0.9976. The best-fit slope is 12 ms/gadget.

WSPL for the Community Page is measured to be 12 milliseconds per Gadget Web Services integrated.

The data shows that over the range tested, the response time difference vs. number of gadget webs services per Community Page over the baseline is linear, and that each additional gadget per Community Page causes a 12 milliseconds response time increase. This value is the same as the corresponding value for My Page to within error estimates.

This means that even for ten gadgets per Community Page, the response time increase due to aggregating additional dynamic web services is negligible, and the same as for the My Page.

Tests Varying the Number of Gadget Web Services Concurrently Accessed for the My Page

Average gadget response time was set to 500 milliseconds. The response time measured for a test run is the base 500 milliseconds for the Gadget Web Service plus a baseline overhead due to the personalized page plus a latency factor depending on the number of Gadget Web Services aggregated per My Page and the number of Gadget Web Services concurrently accessed. The response time differences for different numbers of concurrently accessed web services, for fixed values of the number of Gadget Web Services aggregated per My Page, are data which isolate the latency due to integrating different numbers of Gadget Web Services.

Number of Gadgets per My Page	Number of Concurrently Accessed Web Services	Response Time (ms)	% Total Processor Utilization
2	50	597 +/- 2	25%
2	200	597 +/- 2	21%
2	1000	599 +/- 3	22%
5	50	630 +/- 2	33%
5	200	628 +/- 2	34%
5	1000	629 +/- 2	35%
10	50	686 +/- 2	47%
10	200	690 +/- 3	48%
10	1000	687 +/- 3	49%

Table 4: Data for Response Time and %Total Processor Utilization Dependence on Number of Concurrently Accessed Gadget Web Services for different numbers of Gadgets per My Page

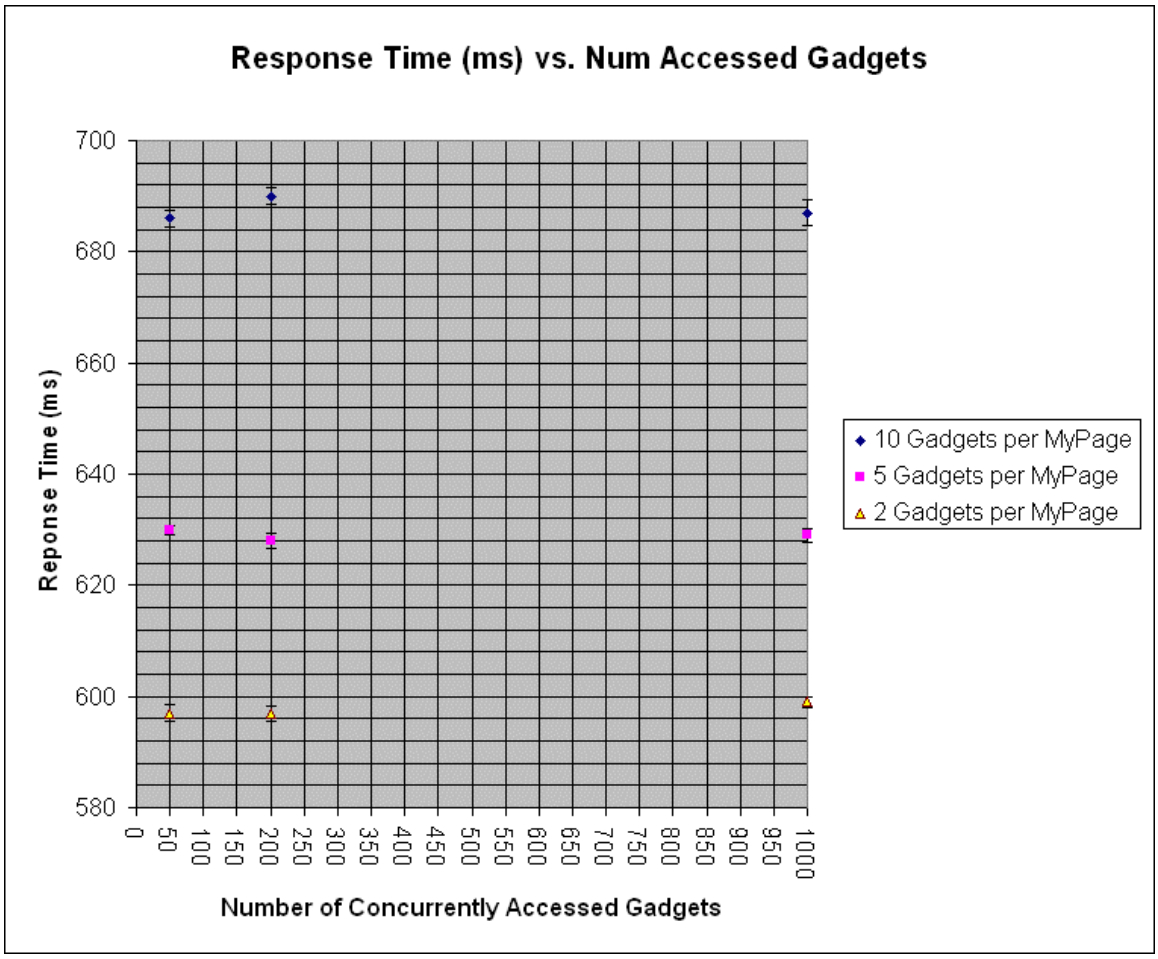

Figure 5: Response Time vs. Number of Concurrently Accessed Gadgets for Three Fixed Values of Number of Gadget Web Services aggregated per My Page

Response time differences for each data series are zero to within error estimates.

The response time differences for tests with 50, 200, and 1000 concurrently accessed web services, for each value of the number of gadget web services per My Page, are the same to within error estimates. That is, WSCAL[2], WSCAL[5], and WSCAL[10] are zero milliseconds per gadget web service concurrently utilized and independent of *n*. The data shows that, for a fixed number of gadgets per My Page and a fixed load, the My Page response time does not depend on the number of concurrently accessed Gadget Web Services.

This means that at least 1000 Gadget Web Services than can be added to the portal system without impacting performance, so that the Portal system can scale for future expansion without modification.

Tests Varying the Number of Gadget Web Services Concurrently Accessed for the Community Page

Average gadget response time was set to 500 milliseconds. The response time measured for a test run is the base 500 milliseconds for the Gadget Web Service plus a baseline overhead due to the personalized page plus a latency factor depending on the number of Gadget Web Services aggregated per Community Page and the number of Gadget Web Services concurrently accessed. The response time differences for different numbers of concurrently accessed web services, for fixed values of the number of Gadget Web Services aggregated per Community Page, are values which isolates the latency due to integrating different numbers of Gadget Web Services.

Number of Concurrently Accessed Web Services	Response Time (ms)	% Total Processor Utilization
50	740 +/- 3	57%
200	743 +/- 3	59%
1000	750 +/- 1	60%

Table 5: Data for Response Time and %Total Processor Utilization Dependence on Number of Concurrently Accessed Gadget Web Services for Ten Gadgets per Community Page

The response time differences for tests with 50, 200, and 1000 concurrently accessed web services are the same to within error estimates. That is, WSCAL[10] for the Community Page is zero—the average is 0.01 milliseconds per gadget web service concurrently utilized. The data shows that, for ten gadgets per Community Page and a fixed load, the Community Page response time does not depend on the number of concurrently accessed Gadget Web Services. The data shows that the Community Page performs similarly to the My Page.

This validates that at least 1000 Gadget Web Services than can be added to the portal system without impacting performance, so that the Portal system can scale for future expansion without modification.

Discussion of the Dependence of Response Time on Caching

Average gadget response time was set to 500 milliseconds. The response time measured for a test run is the base 500 milliseconds for the Gadget Web Service plus a baseline overhead of the personalized page plus the latency due to the number of un-cached Gadget Web Services aggregated per My Page, plus the latency due to the number of cached Gadget Web Services aggregated per My Page. The response time differences for different numbers of un-cached gadget web services per My Page are data which isolate the latency due to integrating cached vs. un-cached Gadget Web Services.

Number of Un-Cached Gadgets per My Page	Number of Cached Gadgets per My Page	Average My Page Response Time (ms)	% Total Processor Utilization
0	10	83 +/- 1	39%
1	9	633 +/- 2	37%
2	8	646 +/- 2	43%
5	5	661 +/- 3	45%
7	3	672 +/- 2	45%
10	0	678 +/- 2	45%

Table 6: Average My Page Response Time Data for Ten Fixed Gadgets per My Page By the Number of Un-cached Gadgets per My Page

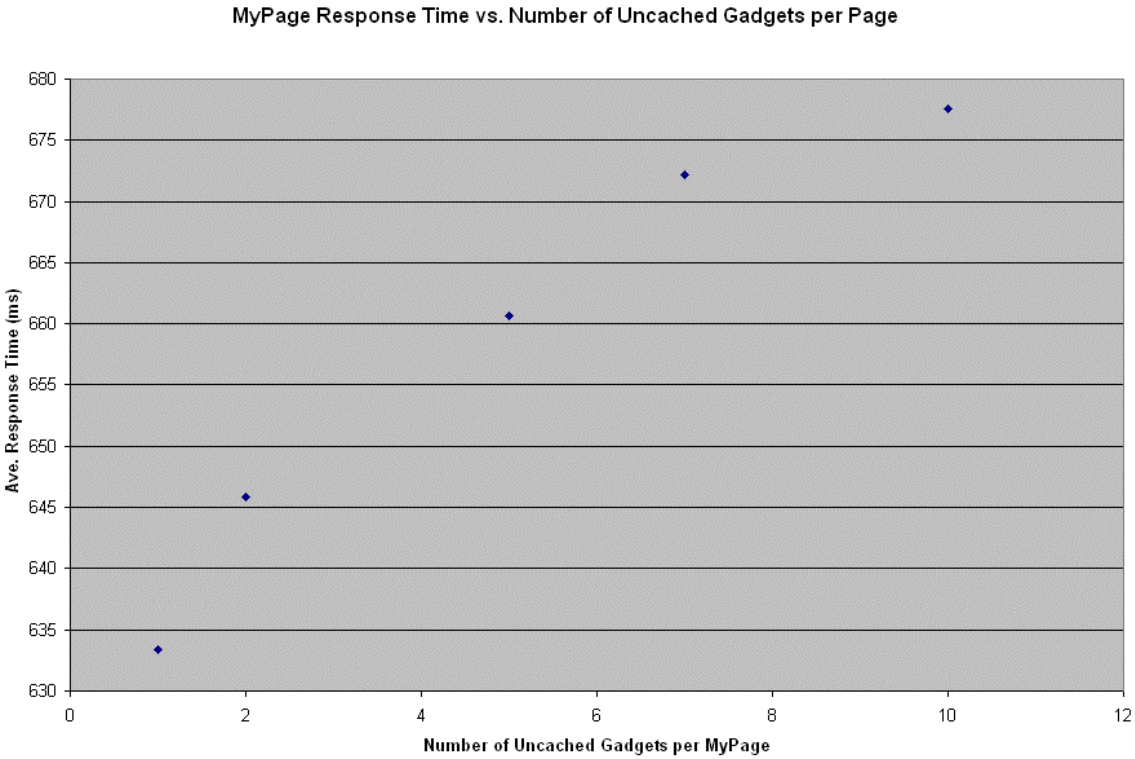

Figure 6: My Page Response Time vs. Number of Uncached Gadgets per Page

The results from Table 6 are plotted above, except the case of no un-cached gadgets per page.

As long as one gadget is not cached, the response time will be above the 500 milliseconds response time of the WaitGadget. However, when all gadgets are cached, one can see that the cache performance cost plus any page overhead performance cost is very small; a page with all cached gadgets responds in less than 100 milliseconds.

The cases tested in previous sections all forced no caching, simulating a worst case performance scenario where all content and services are dynamic. This scenario could happen in real deployments, but is less likely than a mixed case where some gadgets are cached. See Appendix A for more information about caching.

The overall average performance of a real deployment depends on many factors, including the distribution of response times for each user for each un-cached and cached gadget, and the gadget utilization distribution in the profiles of each user personalized page and community. Use of caching is likely to decrease the average response time of the system and increase the scalability of the system in general because there is less processing and fewer slow gadgets are in use, provided the caching strategy is appropriate. Generally deployments will use caching for gadgets which have a long response time or consume large amounts of machine resources on the gadget server. However, there will be important gadgets which must be dynamic, such as e-mail gadgets.

The data in Table 6 shows that the total response time difference between the case with one un-cached gadget per page and the case with ten un-cached gadgets per page is about 45 milliseconds. This shows that the Parallel Portal Engine is efficient in that dynamic content can be delivered almost as quickly as cached content.

The data shows that if all gadgets have the same response time, integrating dynamic content rather than static cached content has a negligible effect on performance.[8] However, if the distribution of gadget response times for different users is large, then individual peak response times could become unacceptable even though the average response time for a page is small. This distribution will be very deployment- and gadget- specific.

The tests conducted illustrate the efficiency of the Parallel Portal Engine, and the Parallel Portal Engine cache, which execute in parallel. The results show that because all requests, both dynamic and cached, are executed in parallel, the overall response time for any page is at most a few hundred milliseconds above that of the gadget with the largest latency, where cached gadgets have a very short latency. While the exact performance values on real world systems cannot be clearly characterized from this data, the results show that the Portal responds efficiently to changes in how much content is cached, such that the real performance impact is mostly independent of the Portal; it depends on the Gadget Web Services used and the caching strategy followed.

Discussion of the Dependence of Response Time on the Fixed Value of Load Used

The scaling under load of the tested system is limited by Portal processor utilization. For loads leaving the processor utilization below 100%, the response time is sub-second and the increase over baseline in response time is at most (near peak) about 200 milliseconds. Additionally, the curve is non-linear, such that response times for loads causing less than about 70% processor utilization are less than about 100 milliseconds. This means that response time differences may change slightly when running the test at a different fixed load, but that in an absolute sense response times will always remain sub-second and the relative increases in response times due to different parameter settings will still be on the order of 10 milliseconds, not 100 milliseconds.

[8] This is not meant to suggest that caching is not important in the portal system architecture, merely to suggest that caching alone will not improve real-world performance for end-users so long as some gadgets are not cached. Use of caching can lower the overall average response time, depending on the gadgets used and the details of the caching model. Caching will also significantly reduce load on the Gadget Servers as well as network traffic, two factors that are vital to consider for scalable portal architectures.

Conclusion

The metric WSPL, which measures the performance impact of aggregating many gadgets on a personalized page, was measured to be 11 milliseconds per Gadget Web Service integrated. This shows that the Parallel Portal Engine efficiently aggregates Gadget Web Services, and that variations in real world user page performance due to users changing the number of Gadget Web Services viewed per page is negligible. This behavior can be contrasted with that of portals using serial Gadget Web Service processing, where WSPL is expected to be on the order of one half second, reflecting absolute response times on the order of five seconds. The portal will thus scale predictably and performance will be stable as more Gadget Web Services are integrated into the system.

The metric WSCAL[*n*], which measures the performance impact of scaling the number of accessed Gadget Web Services, was measured to be zero milliseconds per gadget web service concurrently utilized, within error estimates, for *n* values 2, 5, and 10. The data shows that the Plumtree Portal can scale to integrate at least 1000 Gadget Web Services with unchanged performance. The Plumtree Portal system is thus easily adaptable to meet expanding business needs, with no change to the Portal Platform system and no performance impact.

The effects on the conclusions above of changes in the number of cached Gadget Web Services per page can be understood. The response time change of the personalized Portal pages due to aggregating additional Gadget Web Services is negligible, whether the gadgets are cached or un-cached, for a fixed set of average gadgets. The measurements made isolate the performance impact due to the Portal Platform from that due to deployment caching strategy and Gadget Web Service particulars, showing that the portal server itself is very efficient in managing cached and un-cached web service aggregation.

The dependence on load per Portal server of personalized Portal page response time is small—less than 200 milliseconds overall variation—for Portal servers which are not saturated. The conclusions based on measurements of WSPL and WSCAL[*n*] can be extended to cover a wide variety of real-world deployment scenarios because the Parallel Portal Engine efficiently aggregates cached and un-cached content, and efficiently responds to increased load.

Appendix A: Portal Side Remote Gadget Web Service Caching

The metrics measured use no caching of Web service content, as discussed above. In a real-world case, some fraction of the remote Gadget Web Services utilized will be cached. Tests showed that for the Plumtree Portal, caching does not significantly alter the portal personalized page response times as long as there are some gadgets that remain uncached, so the metrics measured are a good indication of real world portal performance.

Consider the possible caching models for Portals integrating remote Gadget Web Services:

- Portal content caching (PCC), which can be of two types: global portal content caching (GPCC) or personal portal content caching (PPCC). Utilizing GPCC means that no requests of the Web service for any content after an initialization request are required. Utilizing PPCC means that a secure personalized cache is maintained by the portal. No requests of the Web service for any content after an initialization request are required for a particular set of personalization settings. Note that user authentication personalization can be included in this definition, but is not necessary. GPCC and PPCC exhibit identical characteristics in the model used for the benchmark. The net effect of realistic cache hit rates will be reproduced by setting the relative number of Web services utilizing PCC with those utilizing no caching.
- Web service caching (WSC). Utilizing WSC means that the portal always makes a request to the Web service to determine if content has changed. If content has not changed, then cached content is displayed. Note that for the purposes of this benchmark, Web services utilizing WSC will cache all content. The net effects of un-cached responses will be reproduced by setting the relative number of Web services utilizing WSC and those using no caching.
- No caching. Utilizing no caching means that the portal always makes a request of the Web service which always returns content for display.

One can model a distribution of caching models in use by changing the relative number of aggregated remote Gadget Web Services utilizing no caching and the number utilizing and PCC. This is a valid model because gadget side caching would be processed on a server remote from the portal server, and so would behave like a Web service using no caching to the portal server. The overhead on the portal due to making a request and receiving a 304 response (no content change) is negligible for a fast network. Utilizing gadget side caching may allow the gadget to respond more quickly than otherwise, but there are many factors which contribute to the overall latency of the Gadget Web Service. It is reasonable to use the same response time as for uncached gadgets; this models a real-world case of network or gadget application latency, as well as a valid worst case scenario for complicated gadgets.

Designate a distribution of caching models by *n*uc*m*c where *n* is the number of uncached gadgets and *m* is the number of cached gadgets per page.

The dependence of the performance of the portal on the relative number of aggregated remote Gadget Web Services was measured for 10uc1c, 7uc3c, 5uc5c, 2uc8c, and 1uc9c for the Plumtree Portal My Page. These measurements utilized test 10 where the cache settings on the gadget identification object in the Plumtree Portal were set appropriately.

Appendix B: LoadRunner Script Code

Initialization Script. The scenario initializes all vusers with this script before iterations begin.

```
#include "as_web.h"

vuser_init()
{
	// View the Login page for the Portal
	web_url("login.asp",
		"URL=http://ptwebserver3/portal45/admin/login.asp",
		"RecContentType=text/html",
		LAST);

	lr_think_time(3);

	// Create a parameter to hold the userid query string
	web_create_html_param("UserIDQS",
		"mypage.asp?",
		"&");

	// Login to the portal as a distinct user; it will re-direct to the user's MyPage and
	// set the web parameter value
web_submit_form("dologin.asp",
		ITEMDATA,
		"name=UserName",
		"value={UserName}",			//{UserName} is parameterized
		ENDITEM,
		"name=Password",
		"value=plumtree",			//All users have fixed passwords
		ENDITEM,
		"name=selAuthSource",
		"value=10",				//Set authsource to test group name
		ENDITEM,
		"name=Remember",
		"value=<OFF>",
		ENDITEM,
		LAST);

	return 0;
}
```

Main Action Script. This is the action the scenario iterates.

```
#include "as_web.h"


Action1()
{

// Use the first URL in the command below to test the MyPage; comment out the second URL
// Use the second URL in the command below to test the Community Page; comment out the first
// URL

        // Request the user's MyPage or Community Page
        web_url("mypage.asp",
                "URL=http://ptwebserver3/portal45/mypage/mypage.asp?{UserIDQS}&",
//"URL=http://ptwebserver3/portal45/communities/community.asp?{UserIDQS}&I
        ntCommunityIndex=0&intCurrentPageIndex=0",
                "TargetFrame=",
                "RecContentType=text/html",
                LAST);

        // Verify that the string "error" does not appear in the response
        web_find("ErrorSearch",
                "expect=notfound",
                "what=error",
                LAST);

        // Verify that the string "timeout" does not appear in the response
        web_find("TimeoutSearch",
                "expect=notfound",
                "what=timeout",
                LAST);

        return 0;
}
```

End Script. This is run after the test is completed.

```
#include "as_web.h"


vuser_end()
{

         //the following code logs the virtual user of the portal.
        web_url("dologoff.asp",
                "URL=http://ptwebserver3/portal45/admin/dologoff.asp?{UserIDQS}&",
                "TargetFrame=",
                "RecContentType=text/html",
                LAST);

          return 0;
}
```